Bit Twiddling in C

Karkal Prabhu and Naga Kandasamy
ECE Department, Drexel University

In this assignment, you will write C functions that perform some useful bit-level operations. Please submit original work. You can find detailed submission instructions towards the end of this document. The assignment is worth twenty points.

Introduction to Bit Operators in C

One of the advantages of using the C programming language over others is its ability to manipulate contents of memory locations using bit operators such as bit-by-bit AND (&), bit-by-bit OR (|), and bit-by-bit XOR ($\wedge$). The left and right shift operators, $<<$ and $>>$, move bits to the left and to the right by the specified amount, respectively. In an *arithmetic shift*, the bits that are shifted out of either end are discarded; so some bits will be shifted out of the register at one end, while the same number of bits are shifted in from the other end.

The left-shift operator, $<<$, causes the bits to be shifted to the left by the number of positions specified. The bit positions that have been vacated by the shift operation are zero-filled, that is, zeros are shifted in from the right. A left arithmetic shift by n bits is equivalent to multiplying by 2^n provided the resulting value does not overflow. For example let consider an unsigned 8-bit integer `x`.

```
x = 11010100            // Decimal 212 when x is unsiged 8-bit number
x << 1 = 110101000      // 212 x 2^1  = 424
x << 0 = 11010100       // 212 x 2^0 = 212; no shift
x << 4 = 110101000000   // 212 * 2^4 = 3392
```

The right-shift operator, $>>$, causes the bit pattern to be shifted to the right by the number of positions specified. For unsigned numbers, the bit positions that have been vacated by the shift operation are zero-filled. Again, considering an unsigned 8-bit integer `x`.

```
x = 11010100            // Decimal 212 when x is unsiged 8-bit number
x >> 2 = 00110101       // 212/2^2 = 212/4 = 53
x >> 7 = 00000001       // 212/2^7 = 212/128 = 1
x >> 8 = 00000000       // 212/256 = 0
x >> 0 = 11010100       // 212/2^0 = 212; no shift
```

For signed numbers, the sign bit, which is the most-significant bit (MSB) in the two's complement representation, is used to fill the vacated bit positions. In other words, if the number is positive, 0 is used, and if the number is negative, 1 is used to preserve the sign of the operand. A right arithmetic shift by n of a two's complement value is equivalent to dividing by 2^n and rounding toward negative infinity.

```
x = 11010100            // Decimal -44 when x is a signed 8-bit number
x >> 2 = 11110101       // -44/2^2 = -44/4 = -11
x >> 7 = 11111111       // -44/2^7 = -44/128 = -1
x >> 3 = 11111010       // -44/2^3 = -44/8 =  = -6
x >> 0 = 11010100       // -44/2^0 = -44; no shift
```

In a *logical shift*, zeros are shifted in to replace the discarded bits. Therefore the logical and arithmetic left-shifts are exactly the same. However, as the logical right-shift inserts the value 0 into the most significant bit, instead of copying the sign bit, it is ideal for unsigned binary numbers, while the arithmetic right-shift is ideal for signed two's complement binary numbers.

The following functions show how to set, reset, and toggle a bit at a given position using the OR, AND, and XOR functions, and shift operators.

```
/* Function sets the specified bit position within x to 1.
   The least-significant bit (LSB) is bit position 0. */
uint setBit(uint x, uint position)
{
    uint mask = 1;
    return (mask << position) | x;
}

/* Function resets the specified bit position within x to 0. */
uint resetBit(uint x, uint position)
{
    uint mask = 1;
    return (~(mask << position)) & x;
}

/* Function toggles the specified bit position within x to 0. */
uint toggleBit(uint x, uint position)
{
    uint mask = 1;
    return (mask << position) ^ x;
}
```

Finally, it is important to understand the difference between logical and bitwise operators.

- A logical operator, such as AND (&&), OR (||), and NOT (!), is global and returns a binary result (0 or 1).
- A bitwise operator, such as AND (&), OR (|), XOR ($\wedge$), and complement ($\sim$), operates on each bit individually and returns an integer.

Programming Problems

You have been provided with some skeleton code in the project called `bit_operations` that can be downloaded from BBLearn. The `main.c` file is as follows.

```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <stdint.h>
#include <stdbool.h>
#include "bit_operations.h"
#include "uart_functions.h"

int main(void)
{
    MAP_WDT_A_holdTimer();
    initUART();
    writeString("Established communication with the board");

    uint8_t v1 = 0xF7;    /* Hex representation of integer */
    int count = countBits(v1);
    writeInt(count);    /* Display result on terminal */

    uint8_t v2 = 0x0A;    /* Hex representation of integer */
    int status = isPowerOfTwo(v2);
    writeInt(status);

    uint8_t v3 = 0x56;
    uint8_t rv3 = rearrangeBits(v3);
    writeHex((int)rv3);

    while (1) {
    }
}
```

Using the provided code as the starting point, complete the following functions on the MSP432, for five points each.

- **Counting bits.** Complete the function `countBits()` in the `bit_operations.c` file that accepts an unsigned 8-bit integer value as the input argument, and counts and displays the number of bits set in that integer. For example, the number 0xF7 (decimal), which is 11110111 (binary), has seven bits set.
- **Checking for Power of Two.** Complete `isPowerOfTwo()` in the `bit_operations.c` file that takes an unsigned 8-bit integer as input and returns 1 if the number is a power of two, 0 otherwise. Note that the value 0 is not a power of two. Display the answer on the terminal.
- **Rearranging Bits.** Complete the function `rearrangeBits()` that takes the '1' bits in an unsigned 8-bit integer and shifts them to the left end. For example, 0x56 = 01010110 (binary) would become 0xf0 = 11110000 (binary).

- Finally, complete the function `WriteHex()` in `uart_functions.c` that takes as input an integer and displays the corresponding hexadecimal representation on the terminal.

Submission Instructions

Your C code must be clearly written, properly formatted, and well commented for full credit. A good synopsis of the classic book, *Elements of Programming Style* by Kernighan and Plauger, is available at `en.wikipedia.org/wiki/The_Elements_of_Programming_Style`. Another good reference is Rob Pike's notes on *Programming in C*, available at `www.maultech.com/chrislott/resources/cstyle/pikestyle.html`.

Once you have implemented all of the required features described in this document submit your code by doing the following:

- Run `Clean Project` to remove the executable and object files from the project folder. We must be able to build your project from source and we don't require your pre-compiled executables or intermediate object files. **If your code does not at the very least compile, you will receive a zero.**
- Zip up your project and upload the zip file using the Blackboard Learn submission link found on the course website.